

아래는 “드론은 수동(teleop)으로 이동하며 Gazebo에서 얻은 x,y 좌표를 GUI에 전달하고, GUI가 그 좌표를 터틀봇3의 Nav2에 전달하여 해당 지점으로 자율주행하게 하는” 과정을 구현하는 개략적인 방법입니다. 이미 GUI를 만들어두셨다고 하니, 드론 좌표 획득 → GUI 전달 → 터틀봇 Nav2 Goal 로 이어지는 ROS2 노드 설계와 좌표계 관련 유의사항에 초점을 맞추겠습니다.

---

## 1. 드론의 위치(x,y) 획득

### (1) 드론 Pose 토픽 확인

- 일반적으로 드론이 Gazebo에서 동작하면 다음과 같은 토픽 중 하나를 통해 위치/자세를 얻을 수 있습니다.
  - /odom (Odometry 메시지)
  - /amcl\_pose (SLAM/Localization이 붙은 경우)
  - /tf (TF transform 중 map -> base\_link 등)
  - 드론 플러그인에서 별도로 퍼블리시하는 토픽 (예: /simple\_drone/odom, /simple\_drone/pose 등)
- Odometry(nav\_msgs/Odometry) 형태라면 msg.pose.pose.position.x, msg.pose.pose.position.y 로 x,y 좌표를 얻을 수 있습니다.
- TF를 이용한다면 tf2\_ros.TransformListener 등을 통해 현재 map 프레임에서 드론의 base\_link (또는 odom -> base\_link) 변환을 추적할 수 있습니다.

### (2) 드론 Pose Subscriber 노드 구현

- 간단히 rclpy 노드로 /drone\_pose(예시)에 구독하고, 콜백에서 x,y 좌표를 변수로 저장해둔다

## 2. GUI에서 좌표를 입력받아 터틀봇 Nav2에 Goal 전달

### (1) Nav2에서 Goal을 주는 방식

- Nav2는 ROS2 Action을 통해 목표 지점(NavigateToPose)을 받습니다.
- 대표적으로 nav2\_msgs/action/NavigateToPose 액션 서버를 사용

### (2) GUI에서 TurtleBot3로 Goal 전송 로직

- 이미 GUI가 있다고 하셨으니, GUI가 (1) 직접 Action Client를 열어 Nav2에 Goal을 보내거나, (2) 중간 노드에 x,y를 서비스/토픽으로 전달하면, 그 노드가 Action Client로 Nav2 Goal을 전송하는 구조가 가능함

### 3) 드론 좌표계 vs. TurtleBot3 지도 좌표계 일치 여부

- **중요:** 드론이 보는 좌표(예: world 프레임)와 TurtleBot3가 Nav2에서 쓰는 map 프레임이 다를 수 있습니다.
- 만약 Gazebo의 world 프레임과 TurtleBot3의 map 프레임이 동일(또는 정합)하다면, 드론의 x,y를 그대로 Nav2에 전달해도 문제없습니다.
- 하지만 **오프셋**(시작 위치가 다르다거나, 드론과 터틀봇이 서로 다른 SLAM origin을 가질 때)이 있을 경우, x,y를 보정/변환해주어야 합니다.
  - 예: TF2를 이용해 world -> map 변환을 구한 뒤, 드론 좌표를 map 좌표로 변환

지피티야 그러면 드론의 좌표를 획득하고 gui에 전달하려면 드론 pose 토픽을 확인해야하고 드론 pose subscriber노드를 구현해야하는데 내가 너에게 준 tree기준으로 어디에다 넣으면 되는거야? 이제부터 그 메커니즘은 알겠으니까 필요한 코드들을 만들어줘

아래 예시는 “드론의 오도메트리(odom) 토픽을 구독하여 x,y,z 좌표를 읽고, 이를 **drone\_position**이라는 새 토픽으로 퍼블리시” 하는 노드를 만드는 과정입니다.

이 노드를 **sjtu\_drone\_control** 패키지 안에 새로 만들어두고, GUI 쪽에서는 이 **drone\_position** 토픽을 구독하거나 서비스/소켓 등으로 받아서 활용하면 됩니다.

---

## 1. 새 노드 파일 생성

경로:

sjtu\_drone-ros2/sjtu\_drone\_control/sjtu\_drone\_control/drone\_pose\_subscriber.py

## 2. setup.py 수정

이 노드를 실행 가능하도록 **setup.py** 에 console script를 등록합니다

드론이 simple\_drone/odom 토픽을 퍼블리시 중인거를 알고 drone\_pose\_subscriber노드가 drone\_position 토픽을 계속 퍼블리시하는걸 알려면 먼저 빌드를 한담에 새로운 터미널에서 setup후 `ros2 launch my_combined_launch combined.launch.py` 으로 드론하고 터틀봇이 같이 나오면 계속 토픽을 퍼블리시하는거고 그다음에 `ros2 run sjtu_drone_control`

drone\_pose\_subscriber으로 확인을 한다음에 ros2 topic echo /drone\_position 명령으로 실제 x,y,z 좌표가 잘 나오는지 확인을 해야하는거지?

응 맞아~

## 프로젝트 개요

- 목표

“드론은 수동(teleop)으로 이동하며 Gazebo에서 얻은 x,y 좌표를 GUI에 전달하고, GUI가 그 좌표를 터틀봇3의 Nav2에 전달하여 해당 지점으로 자율주행하게 하는” 과정을 구현.
  - 주요 구성
    - 드론(Gazebo 상에서 동작)
      - Teleop으로 이동
      - 드론의 좌표를 퍼블리시
    - GUI(test.py)
      - 드론의 좌표를 구독
      - Turtlebot3에 Goal(PoseStamped) 또는 Twist 명령을 전달
      - 카메라(바텀 카메라) 영상 표시
    - Turtlebot3
      - Nav2(또는 직접 Twist)로 자율주행
      - 로봇팔(Manipulation) 장착
- 

## 문제 해결 과정

### 문제점1: 새로 수정한 world가 반영되지 않음

- 상황

`ros2 launch sjtu_drone_bringup sjtu_drone_bringup.launch.py`  
명령어로 Gazebo를 실행했을 때, 수정한 world 파일이 반영되지 않음.
- 원인

`sjtu_drone_bringup.launch.py` 내부에서 사용 중인 world 파일 경로가 실제 수정한 파일 경로와 불일치.
- 해결
  - launch 파일 내 world 파일 경로를 실제 수정한 경로와 일치하도록 수정하니 해결.

### 문제점2: warehouse.world 내 모델(박스, mines 폴더) 반영 안 됨

- 상황

warehouse.world에 박스들을 추가했으나, Gazebo에서 해당 모델이 나타나지 않음.
- 원인
  - world 파일에서 모델을 불러올 때, `model://mines` 경로를 찾지 못함.
  - Gazebo가 mines 폴더 위치를 알 수 없는 상태.
- 해결

- **GAZEBO\_MODEL\_PATH 설정**

```
bash
CopyEdit
cd
/home/seungrok/ros222_ws/src/sjtu_drone-ros2/sjtu_drone_description/
worlds
export
GAZEBO_MODEL_PATH=$PWD/warehouse_mines_v0.0:$GAZEBO_MODEL_PATH
```

- 단, ros2 launch 시에는 셸 환경이 달라 적용이 안 될 수 있음.

- **~/.gazebo/models 폴더 사용**

```
bash
CopyEdit
mkdir -p ~/.gazebo/models
cp -r warehouse_mines_v0.0_w_color/mines ~/.gazebo/models
```

- Gazebo가 기본적으로 ~/.gazebo/models를 검색하므로, mines 폴더를 그 안에 넣어주면 인식 가능.

### 문제점3: 드론 패키지에 Turtlebot3를 함께 띄우고 싶음

- **상황**

기존에 드론만 올라가는 패키지(sjtu\_drone\_bringup)에서 Turtlebot3도 함께 스폰하고 싶음.

- **해결**

- **combined.launch.py**를 새로 작성
- 드론 관련 launch와 터틀봇3 관련 launch를 동일 워크스페이스에서 실행할 수 있도록 병합함.
- setup.py에서 entry\_points 등을 수정하고, 빌드 후 사용.

### 문제점4: Turtlebot3 모델에 Manipulation이 달리지 않음

- **상황**

turtlebot3\_manipulation 패키지를 써서 로봇팔을 올리려 했으나, 실제 Gazebo 상에는 기본 모델만 등장.

- **원인**

- robot\_state\_publisher가 로봇팔이 정의된 URDF를 퍼블리시해야 하는데, 기본 Turtlebot3 URDF만 사용 중.

- **해결**

- **robot\_state\_publisher**가 turtlebot3\_manipulation.urdf.xacro를 사용하도록 수정
- **spawn\_turtlebot3.launch.py**에서 -file 옵션 대신 -topic 옵션을 사용해, robot\_description 토픽을 통해 스폰하도록 변경.

## 문제점5: Turtlebot3가 날아다님(드론처럼 떠버림)

- 상황  
드론 대신 Turtlebot3가 공중에 뜨는 이상 현상.
- 원인
  - 두 로봇이 같은 토픽(cmd\_vel 등)을 사용하여 충돌 발생.
- 해결
  - 네임스페이스 분리
    - 드론: /drone/cmd\_vel, /drone/odometry, ...
    - 터틀봇3: /turtlebot3/cmd\_vel, /turtlebot3/odom, ...
  - launch 파일에서 각각의 로봇에 고유 네임스페이스 지정.

## 문제점6: 드론 초기 위치 설정

- 상황  
Gazebo 상에서 드론과 터틀봇3의 시작 위치를 조정하고 싶음.
- 해결
  - spawn\_drone.launch.py, spawn\_turtlebot3.launch.py 등의 인자를 수정해 초기 좌표를 원하는 위치(예: -1.013570, 3.022503, -0.000001)로 설정.

## 문제점7: 드론 초기 위치를 바꾸니 teleop이 되지 않음

- 상황  
드론이 URDF의 gazebo 플러그인에서 /simple\_drone/cmd\_vel, /simple\_drone/takeoff 등을 사용하도록 되어 있는데, teleop.py는 /cmd\_vel, /takeoff 등을 퍼블리시.
- 해결
  - teleop 노드(teleop.py, teleop\_joystick.py)에서도 절대 토픽 (/simple\_drone/cmd\_vel, etc.) 사용하도록 변경해 네임스페이스 일치.

## 문제점8: Build 에러 발생

- 상황  
CMakeLists.txt에서 worlds 폴더를 설치하려고 할 때 경로가 달라 빌드 오류.
- 해결
  - CMakeLists.txt 내 install() 구문 수정
  - 올바른 경로 지정 후 다시 빌드해 해결.

## 문제점9: Turtlebot3가 드론 위치 대신 test.py 내 고정 좌표로만 이동

- 상황  
“제거 시작” 버튼을 누르면 test.py가 /goal\_pose 퍼블리시 → Turtlebot3가 움직여야 하는데, 실제 드론 좌표가 아닌 고정 좌표로 이동.
- 원인
  - test.py 로직에서 드론 좌표를 제대로 활용하지 않고, 하드코딩된 좌표로 이동 명령을 주고 있었음.
- 해결
  - test.py에서 /drone\_position 콜백을 통해 받은 self.drone\_x, self.drone\_y를 실제 goal에 반영하도록 수정.
  - 불필요한 수동 Twist 명령 제거(내비게이션 스택을 사용하는 경우 충돌 가능).

## 문제점10: Manipulation이 땅 아래로 박혀 있음

- 상황  
로봇팔 모델이 Gazebo 바닥 아래에 잠겨 있는 현상.
- 원인
  - manipulation을 스폰하는 launch 파일이 다르고, 초기 Joint/URDF 설정이 뒤엉킴.
- 해결
  - turtlebot3\_manipulation\_bringup/gazebo.launch.py와 combined.launch.py의 설정이 달랐음 → 초기 Joint 세팅 통일.
  - 필요한 URDF 파라미터를 병합하여 일관된 로봇팔 초기 상태 설정.

## 문제점11: Turtlebot3가 목표 지점으로 정확히 가지 않고 엉뚱한 곳으로 이동

- 원인
  - /drone\_position이 갱신되지 않거나, 잘못된 토픽을 구독 중.
  - test.py에서 goal\_pose 퍼블리시 후 불필요한 Twist 명령을 함께 보내거나, 프레임 설정이 어긋남(‘world’, ‘odom’ 등).
- 해결
  - 드론 위치를 퍼블리시하는 노드를 확인, /drone\_position에 올바른 데이터 전달.
  - test.py에서 Goal 메시지의 frame\_id, Twist 명령 중복 여부를 제거/조정.

## 문제점12: 로봇이 이제 아예 움직이지 않음

- 상황  
Goal을 퍼블리시해도 Turtlebot3가 반응하지 않음.
- 원인1: 내비게이션 스택 미사용
  - /goal\_pose를 구독할 노드가 없으므로 PoseStamped 보내도 로봇이 안 움직임.
- 원인2: 내비게이션 스택 사용하지만, 토픽/액션 불일치

- Nav2는 /navigate\_to\_pose 액션을 사용. 단순 토픽 퍼블리시로는 동작 안 함.
- 해결
  - 직접 Twist 명령
    - test.py에서 드론 좌표까지의 경로를 직접 계산해 /cmd\_vel 퍼블리시.
  - Nav2 액션 활용
    - /navigate\_to\_pose 액션 서버에 Goal을 보내도록 test.py를 수정.

### 문제점13: tf\_transformations에서 numpy 1.20 이상 버전으로 인해 np.float 오류 발생

- 상황
 

“AttributeError: module 'numpy' has no attribute 'float'” 에러.
- 해결
  - test.py 상단에 다음 코드 추가:
 

```
python
CopyEdit
import numpy as np
np.float = float
```
  - tf\_transformations가 내부적으로 np.float를 호출할 때, 이를 직접 float로 매핑해 해결.

## 결론

위와 같은 문제들을 하나씩 해결해나가며 드론과 Turtlebot3를 하나의 Gazebo 환경에서 동시에 띄우고,  
드론 좌표를 기반으로 Turtlebot3가 이동하도록 구현할 수 있었습니다.

핵심 정리:

1. 워크스페이스 구조 및 경로를 꼼꼼히 확인 (world 파일, 모델 폴더 등).
2. 네임스페이스 분리로 토픽 충돌 방지.
3. URDF & 로봇팔 초기화 일관성 유지.
4. Nav2 또는 Twist 직접 제어 여부를 명확히 하여 Goal 퍼블리시 로직 설계.
5. tf\_transformations & numpy 버전 호환성 문제 해결( np.float → float ).

이와 같은 과정을 거쳐 최종적으로 “드론은 teleop으로 이동, gazebo에서 얻은 x,y를 GUI로 전달, GUI가 Turtlebot3에 Goal 명령”의 파이프라인이 완성되었습니다.